Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 63 (2015) 96 – 103

Procedia Computer Science

The 6th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2015)

Resource sharing in mobile cloud-computing with CoAP

Yi Xue, Ralph Deters*

steven.xui@mail.usask.ca, deters@cs.usask.ca
University of Saskatchewan, Dep. Computer Science, Saskatoon Canada

Abstract

Mobile Cloud-Computing (MCC) is a term introduced by Marc Baccue in 2009[20] that popularized the idea of using cloud-hosted components as a means to overcome the resource-constraints of mobile devices. But as the smartphones and tablets overcame their resource-constraints, the meaning of the term MCC changed. Nowadays MCC is mainly associated with using mobile devices to engage cloud-hosted services and to a lesser extend with combining multiple mobile devices (e.g. cloud of devices). However, as the number of users with *multiple* mobile devices increases there is a growing demand for enabling apps on mobile devices to share hardware and software resources. This in turn leads to questions regarding decentralized interaction, coordination and resource sharing among multiple mobile devices.

This paper focusses on the "horizontal scalability" of apps e.g. the ability to combine multiple mobile devices (executing the same mobile app) into a single compute environment that utilizes all available hardware and software resources in a decentralized manner. One possible approach to achieve this is by designing mobile apps as sets of RESTful micro-services and to allow these services to communicate via low-bandwidth IoT communication protocols. This paper presents the results of our performance evaluations using RESTful micro-services on mobile devices that communicate via the IoT protocol CoAP in different WIFI environments.

Peer-review under responsibility of the Program Chairs

Keywords: Mobile Cloud Computing, CoAP, Device Cloud, IoT

* Corresponding author. Tel.: +1-306-966-2072; fax: +1-306-966-4884.
E-mail address: deters@cs.usask.ca

Peer-review under responsibility of the Program Chairs
doi:10.1016/j.procs.2015.08.317

1. Introduction

As an increasing number of users own more than one mobile devices the need for combining them into a single compute unit arises. This paper focusses on ways to enable an app to "horizontally scale" across multiple mobile devices by allowing it to access all available hardware and software resources within a cloud of mobile devices. It is assumed that each user-owned mobile device has already the app installed and that the mobile app instances communicate directly with each other to avoid centralized communication and coordination structures. Nevertheless, it is assumed that the MCC apps can still engage backend services e.g. cloud-hosted data or application servers.

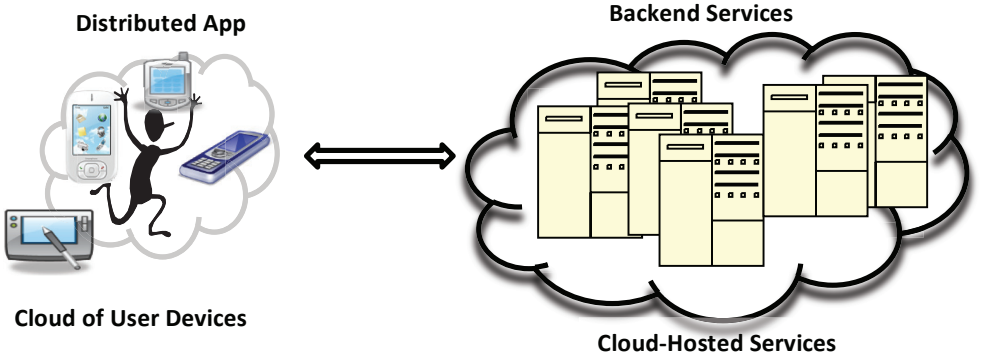

Figure 1: Combing privately owned devices to access cloud services

Wireless communication between devices over short distances has been an active area of research. Using short-range wireless protocols[1,2] it is fairly easy for one mobile device to detect the presence of other nearby mobile devices and establish a connection with any modern mobile OS. Given the widespread adoption of the model-view-controller (MVC) pattern, exchanging (event) messages between the distributed/replicated MVC components has been explored[3,4,5] especially in regards to sharing views across devices. However, it is also possible to achieve a more fine-grained integration of systems by enabling sub-components to interact with each other[6,7] across mobile devices. Interestingly, when communicating with cloud-hosted backend services, the REpresentation State Transfer (REST) pattern [8] has emerged as the de-facto standard since it can be mapped directly to the application level protocol HTTP. The sharing of application and host resources within MCC has been studied primarily in the context of combining the computational resources of mobile devices and offloading. Combining mobile devices into a homogeneous compute platform e.g. via MapReduce[9,10] and/or linking/integrating mobile devices into existing distributed platforms[11,12] has been suggested. However, the costs of providing/managing such a platforms leads to significant overhead and thus the uptake of these concepts has been limited. An alternative to the formation of a platform is the offloading of tasks from resource constrained mobile devices to resource rich devices. VM centric offloading approaches focus on migrating part or parts of a virtual machine to another[13,14,15] . Besides requiring similar VMs these approaches require high-bandwidth and low-latency connections that are difficult to achieve when using short-range wireless protocols. An alternative to the virtual machine centric approach is the partitioning of the application and data (Cuckoo[16] and Weblets[17]). Besides requiring the developer to predefine the part or parts of the application that is to be moved, they still require significant networking resources. More recently, computational REST (CREST) [18,19] has been proposed, but not demonstrated. CREST extends REST[8] with the notion of computational resources (e.g. expression, program, closure, or continuation). This, in turn, allows REST compliant sharing of computational resources.

In this paper we focus on the combination of the IoT protocol CoAP and the RESTful micro-services pattern as an alternative to existing approaches[22,23,24,25,26,27] for seamless resource/component sharing across multiple mobile hosts. We focus on the use of the REST design pattern and low-bandwidth IoT communication protocols as a means to enable components/micro-services of the same app residing on different mobile host to seamlessly communicate.

The remainder of the paper is structured as follows. Section 2 discusses briefly the horizontal scalability of mobile apps. This is followed by an introduction of the CoAP protocol in section 3 and the evaluation of this protocol in section 4. The paper concludes with a summary and outlook.

2. Horizontal Scalability of Mobile Apps

Horizontal scaling (scaling out) refers to the ability of a system to use additional hardware (e.g. nodes) to improve its performance e.g. handle higher loads. Horizontal scalability is most often linked to cloud-services and servers and tends to be rarely used in the context of mobile apps. However, as users begin to own multiple devices the question emerges if an app that resides on multiple mobile devices can be enabled to seamlessly use the resources of N mobile hosts in order to handle higher loads or perform tasks faster/better. In case of an app that follows the MVC pattern, the question arises if this app could "scale out" its viewer e.g. combine multiple screens. Another question is if the models residing on different devices could be combined into one model to access already stored data and avoid costly call to cloud-hosted data-stores. In the context of this paper we view the scaling out of an app as the ability to seamlessly access the various resources in a cloud of user owned mobile devices. By constraining ourselves to the context of a single app we avoid the challenges of cross app interoperability issues e.g. access rights, semantics, etc. We consider the components of the MVC app as collections of resources that can be engaged in a RESTful manner.

3. CoAP – REST over UDP

The CoAP (Constrained Application Protocol) is an IoT protocol that tries to combine publish-subscribe features (e.g. MQTT) with REST. CoAP (RFC7252) was initially proposed by ARM and the University of Bremen (TZI) in 2014. It is a specialized transfer protocol for machine-to-machine communication that has been optimized for constrained nodes and constrained networks. The traditional CoAP nodes have small microcontrollers with limited amounts of ROM and RAM, while constrained networks often have low power and high packet error rates and a typical throughput of 10s of kbit/s. CoAP uses UDP as the transport protocol and its package size varies from 4 bytes to 1024 bytes. It provides a request/response interaction model between application endpoints, supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs and Internet media types. CoAP also can easily interface with HTTP for integration with the Web 0 and also supports encrypted communication. CoAP allows the requesters to subscribe to state-changes in resources by using the "observe" options. Using this option requires the resource to push in case of state-changes the new response to the requester thus implementing a highly customizable pub-sub mechanism.

4. Performance Evaluation

To evaluate the usefulness of CoAP as a means for horizontal scaling of apps we focus on the I/O performance evaluation in different WIFI settings. As mentioned earlier it is assumed that multiple mobile devices share the same app and that the app is implemented as a collection of RESTful micro-services.

4.1 Experimental Setup

The prototype system consists of clients, servers and network infrastructure. Once the server receives a request from the client, it starts collecting the selected data and sends them to the client over a wireless network until the client sends a REST request to cancel the subscription. Both client and server run the same application with the same micro-services but in different modes. In the evaluation, Google Nexus 7 (Android 4.4.3) is used as the server and Samsung GT-P7510 (Android 4.0.4) acts as the client. The set-up is shown in figure 2. The experiments runs over a dedicated network and a public network. The dedicated network uses the TP-Link TL-WR841N wireless router

(802.11b/g/n) as the access point and there are no other devices in the network during the experiments. For the public network, a Starbucks wireless network was chosen. All devices in all experiments have optimal signal strength and there is no background application on neither of the two devices.

Figure 2: Experimental Setup

Resource discovery is an important mechanism in IoT communication protocols. For *discovery*, the clients send a GET request to a special URI (./well-known/core). The URI is part of CoAP protocol standard and supported by all nodes in a CoAP network. When a node receives such request it sends all resources available on this device in the response (e.g. JSON encoded). Once the client decides which resources it wishes to observe (subscribe to state changes), it sends out an "observe" request with chosen id to server. The server adds the client into a subscription list on the chosen resource and once the state changes all subscribers on the list are updated. If the client does no longer wish to be informed about state changes of a resource, it send out a REST request to the server to cancel the I/O subscription.

The CoAP *payload size* is evaluated to investigate the relationship between package size and performance and to determine optimal sizes. The maximum payload size in CoAP is 1024 bytes. After a fixed time interval or a state change of the resource, the current state will be sent to the client. Since these experiments are designed for performance evaluations, it is important to determine the upper boundary of the throughput. The lower boundary is obviously 0 and the upper boundary depends not only on network latency but also on the server request processing time. Sending intervals in all tests is 0, which means that there is no delay added on the server side. To understand how different types of wireless networks impact the I/O performance, a dedicated network and a public network were tested. In order to remove bias on individual sampling, we repeated I/O subscription and I/O transfer tests. The subscription test were repeated 30 times and I/O transfer test -- 100 times. Payload sizes were chosen among 16, 32, 64, 128, 256, 512 and up to 1024 bytes. For each payload size, both the subscription and transfer experiment was tested. The set of data collected are:

- I/O packet sequential number
- Payload size in bytes
- Response time for the subscription and data transfer in milliseconds
- Throughput for data transfer KB/s and for subscription number per second

In each experiment, all collected data was recorded in a log file (client and server side). When an experiment was finished, the mean response times were calculated from individual subscription or data transfer response times. Throughput was calculated from the sum of sent I/O data size divided by its total elapsed transfer time.

To evaluate performance results we focused on the following measurement metrics:

- Subscription Throughput
- Subscription Response time
- Data Throughput
- Data Response time

Throughput is measured in how many bytes or number of subscriptions can be sent or made in every second based on different payload size. Response time is measured in milliseconds.

4.2 Evaluation

In Figure 3, the performance results from the experiments are presented. The blue line represents data collected from the dedicated network and the orange line represents data collected from the public network.

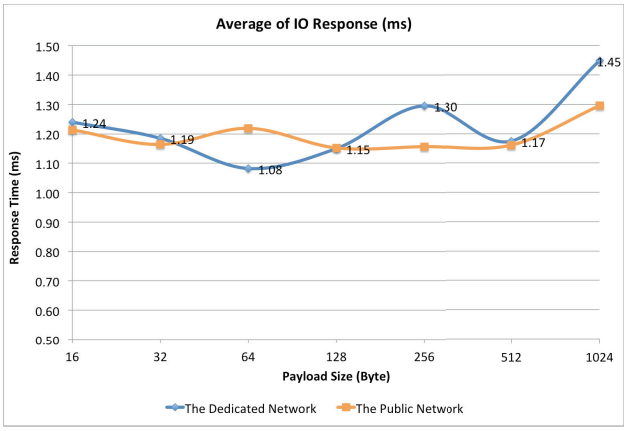

Figure 3: Average IO Data Transfer Response Time

It can be seen that there is no big difference in the data transfer response time with respect to payload size. The response times all fall into a small range, 1.08 to 1.45 ms. Since all response times are very low, any network interference or OS activity can easy cause fluctuation in the response time. Although at some payload sizes the response time does not follow the trend line, the bigger the payload size the bigger the response time.

The data transfer rate grows along payload size and the throughput peaks at ca. 800 KB/s on payload size 1024 bytes for both networks (see Figure 4). This is as expected, since a bigger payload size has a higher I/O throughput. Closely looking at the data, we can see that the I/O throughput approximately doubles as the payload size doubles.

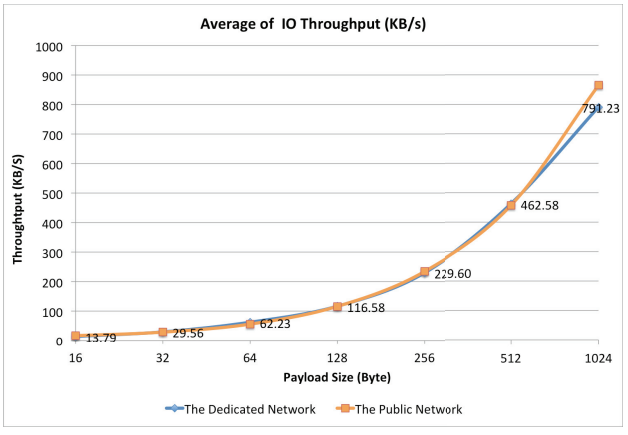

Figure 4: Average IO Data Transfer Throughput

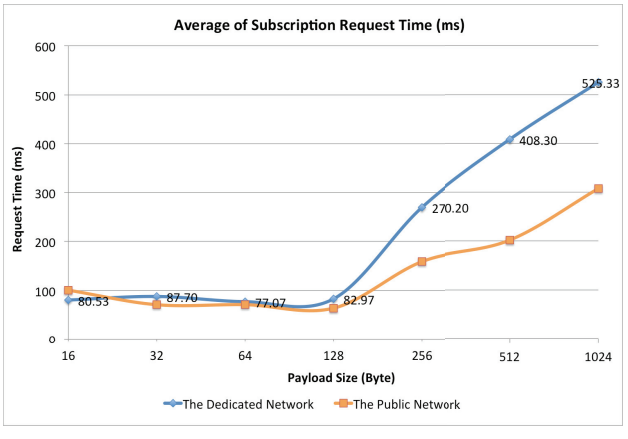

Figure 5: Average Subscription Response Time

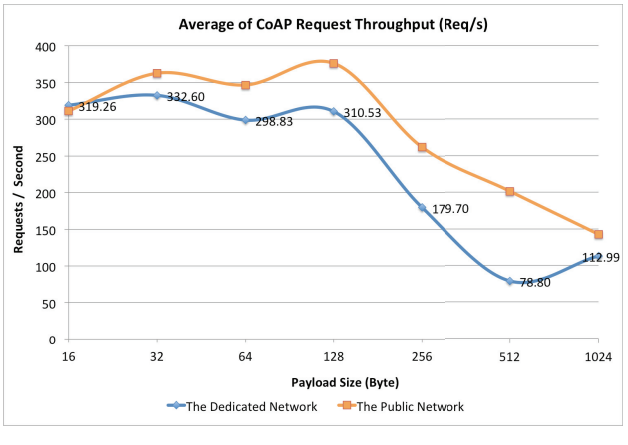

Figure 6: Average Subscription Throughput

For the subscription response time it can be observed that until 128 bytes payload size the response time is low at less than 100ms (see Figure 5). However, starting with 256 bytes payload, the response time increases rapidly. The response time is subject to payload size because it measures the elapsed time including the time that the first data is received. After 128 bytes payload, the doubled payload size introduces extra overhead to finish the request. This extra overhead could come from memory allocation, and/or network transfer. And the "magic" number 128 bytes might also be OS-related because the default network buffer is small on a mobile device. If the application is sensitive to subscription request time, we need to choose a payload size less or equal to128 bytes.

In figure 6, the results echo previous observations although a bigger fluctuation is visible. 128 bytes payload still is the "magic" turning point when the throughput declines. The throughput on the public network is higher than the dedicated network but this is expected, since its response time is faster than the other one. In general, there is no significantly different behaviour between two types of networks.

The I/O performance in both network settings shows response times of less than 1.5ms and a maximum throughput around 800KB/s at 1024 bytes payload size (maximum). The subscription evaluation indicates that 128 bytes payload size is a turning point. Any bigger payload size results in longer response times and lower throughput. In both network settings, we are able to get least 78 subscription requests per second and more than 330 requests per second. The only performance difference between the two networks is that in the public network, we get a bit better performance in terms of response time and throughput. The major factors contributing to these difference are most likely the different routers used in the networks. The router used in the dedicated network is a low-cost router. Although the exact model used in the Starbucks coffee-shop network is unknown, it is assumed to be a high-end Cisco router. High-end routers always use faster processors, more internal memory and more efficient scheduling algorithms that allow faster response times and serve more mobile devices. However, we do not see a big difference between the two network settings, indicating that the performance is more device and OS specific.

5. Conclusions

As an increasing number of users own more mobile devices the need for allowing a mobile app to spread across N devices emerges. This paper focusses on ways to enable apps to "horizontally scale" across multiple mobile devices by accessing all available hardware and software resources. It presents an evaluation of the micro-services pattern in apps for mobile devices that use the IoT communication protocol CoAP. We examine the I/O performance of CoAP in 2 different wireless settings and present the data. The evaluation with two average mobile devices (smartphone & tablet) shows that the maximum throughput is around 800KB/s at the maximum payload size of 1024 bytes. The experiments show that at least 330 subscriptions per second can be achieved. In addition it is observed that the 128 bytes payload size seems ideal in terms of subscriber response time and throughput.

While CoAP uses UDP as a transport protocol, it is possible to use non-IP based transport protocols. Our ongoing work focusses on the use of BLE as means for enabling a single mobile app to scale across multiple mobile devices. Initial results indicate that the feasibility and good performance in replacing UDP with BLE.

References

1. Lee, Jin-Shyan, Yu-Wei Su, and Chung-Chou Shen. "A comparative study of wireless protocols: Bluetooth, UWB, ZigBee, and Wi-Fi." In *Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE*, pp. 46-51. IEEE, 2007.
2. Camps-Mur, Daniel, Andres Garcia-Saavedra, and Pablo Serrano. "Device-to-device communications with Wi-Fi Direct: overview and experimentation." *Wireless Communications, IEEE* 20, no. 3 (2013).
3. Hyo-Joo Han. 2004. *Virtual Teams Combining Mobile Devices with Web-Based Communication on Group Decision Making*. Ph.D. Dissertation. New Jersey Institute of Technology, Newark, NJ, USA.
4. Farshchian, B.A., Divitini, M.: Collaboration Support for Mobile Users in Ubiquitous Environments. In: Handbook of Ambient Intelligence and Smart Environments, pp. 173–199 (2010)
5. Suthers, Daniel D. "Architectures for computer supported collaborative learning." In *Advanced Learning*

Technologies, 2001. Proceedings. IEEE International Conference on, pp. 25-28. IEEE, 2001.
6. Mascolo, Cecilia, Licia Capra, and Wolfgang Emmerich. "Mobile computing middleware." In *Advanced lectures on networking*, pp. 20-58. Springer Berlin Heidelberg, 2002.
7. Gaddah, Abdulbaset, and Thomas Kunz. "A survey of middleware paradigms for mobile computing." *Techical Report, July* (2003).
8. Fielding, Roy Thomas. "Architectural styles and the design of network-based software architectures." PhD diss., University of California, Irvine, 2000.
9. Marinelli, Eugene E. *Hyrax: cloud computing on mobile devices using MapReduce*. No. CMU-CS-09-164. Carnegie Mellon School of Computer Science, 2009.
10. Dou, Adam, Vana Kalogeraki, Dimitrios Gunopulos, Taneli Mielikainen, and Ville H. Tuulos. "Misco: a MapReduce framework for mobile systems." In *Proceedings of the 3rd international conference on pervasive technologies related to assistive environments*, p. 32. ACM, 2010.
11. Chu, David C., and Marty Humphrey. "Mobile ogsi. net: Grid computing on mobile devices." In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pp. 182-191. IEEE, 2004.
12. Kotilainen, Niko, Matthieu Weber, Mikko Vapa, and Juori Vuori. "Mobile Chedar-a peer-to-peer middleware for mobile devices." In *Pervasive Computing and Communications Workshops, 2005. PerCom 2005 Workshops. Third IEEE International Conference on*, pp. 86-90. IEEE, 2005.
13. Satyanarayanan, Mahadev, Paramvir Bahl, Ramón Caceres, and Nigel Davies. "The case for vm-based cloudlets in mobile computing." *Pervasive Computing, IEEE* 8, no. 4 (2009): 14-23.
14. Cuervo, Eduardo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. "MAUI: making smartphones last longer with code offload." In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pp. 49-62. ACM, 2010.
15. Chun, Byung-Gon, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. "Clonecloud: elastic execution between mobile device and cloud." In *Proceedings of the sixth conference on Computer systems*, pp. 301-314. ACM, 2011.
16. Kemp, Roelof, Nicholas Palmer, Thilo Kielmann, and Henri Bal. "Cuckoo: a computation offloading framework for smartphones." In *Mobile Computing, Applications, and Services*, pp. 59-79. Springer Berlin Heidelberg, 2012.
17. Zhang, Xinwen, Anugeetha Kunjithapatham, Sangoh Jeong, and Simon Gibbs. "Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing." *Mobile Networks and Applications* 16, no. 3 (2011): 270-284.
18. Erenkrantz, Justin R., Michael Gorlick, Girish Suryanarayana, and Richard N. Taylor. "From representations to computations: the evolution of web architectures." In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pp. 255-264. ACM, 2007.
19. Gorlick, Justin R. Erenkrantz Michael M., and Richard N. Taylor. "Rethinking Web Services from First Principles."
20. M. Beccue, ABIresearch Report RR-MCC: “Mobile Cloud Computing”, 64 pages, 2009.
21. Z. Shelby, K. Hartke, and C. Bormann. The constrained application protocol (CoAP), 2014.
22. Murphy, Amy L., Gian Pietro Picco, and Gruia-Catalin Roman. "LIME: A coordination model and middleware supporting mobility of hosts and agents." *ACM Transactions on Software Engineering and Methodology (TOSEM)* 15, no. 3 (2006): 279-328.
23. Fernando, Niroshinie, Seng W. Loke, and Wenny Rahayu. "Mobile cloud computing: A survey." *Future Generation Computer Systems* 29, no. 1 (2013): 84-106
24. Wells, George. "A Tuple Space Web Service for Distributed Programming." In *PDPTA*, pp. 444-450. 2006.
25. Barton, John J., Shumin Zhai, and Steve B. Cousins. "Mobile phones will become the primary personal computing devices." In *Mobile Computing Systems and Applications, 2006. WMCSA'06. Proceedings. 7th IEEE Workshop on*, pp. 3-9. IEEE, 2005.
26. Bold Ambition & Our Core, http://news.microsoft.com/ceo/index.html
27. Doolan, Daniel C., Sabin Tabirca, and Laurence T. Yang. "Mmpi a message passing interface for the mobile environment." In *Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia*, pp. 317-321. ACM, 2008.